V&V. El Desafio Final. Verificación y Validación para Y2K.

Excequiel Matamala A.
ematamal@consist.cl

Introducción

A medida que se acerque el advenimiento del año 2000, más y más organizaciones iniciarán la tarea de mejorar sus sistemas computacionales con el objetivo de cumplir con el año 2000. Es muy probable que a inicios de 1999 un gran porcentaje de empresas esté en una etapa de conversión o reemplazo del software para lograr que éste procese correctamente la fecha con lógica de cuatro dígitos para el año. Más aún, serán muchos los casos que para aquella fecha habrán terminado esta etapa de revisión y conversión de millones de líneas de código, millones de registros de almacenes de datos, y millones de interfaces. Lamentablemente, descubrirán entonces, que aquella etapa no era la parte difícil del problema del año 2000 y que la verdadera crisis del 2000 se inicia junto al proceso de verificación y validación del software corregido.

Se esbozará a continuación una metodología de testing para y2k que abarca completamente el proceso de Verificación y Validación de software, desde la planificación del testing en las primeras fases de desarrollo de un proyecto año 2000 hasta la generación de reportes y análisis de los resultados del Testing.

En la figura 1, se exhiben las etapas de la metodología propuesta.

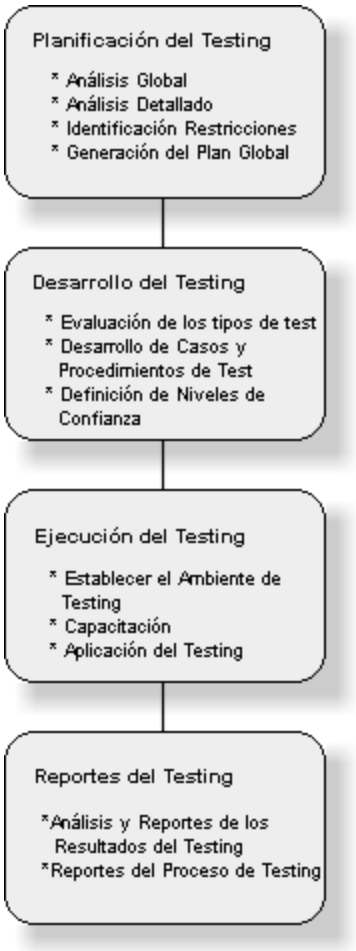

Figura 1. - Metodología de Testing para y2k.

Planificación del Testing

El propósito de la etapa de planificación es realizar un análisis minucioso del sistema y sus componentes hardware y software generando un plan global de test en el que se definen las funciones, los roles, los tipos, técnicas y estrategias de testing a utilizar.

La etapa de planificación debe considerar un Análisis General, un Análisis Específico o Detallado, la identificación de restricciones y la Generación de un Plan Global de Test.

Análisis General.

Antes que nada, se debe determinar la naturaleza del sistema a testear y estimar el esfuerzo total requerido para la planificación, desarrollo y aplicación del testing. Para lo cual se debe analizar el inventario global de los sistemas y el entorno de éstos identificando aquellos componentes que se sometieron a conversión y necesitan ser testeados.

También se debe realizar un análisis de las funciones del sistema, lo que servirá para identificar el dominio de aplicación de éste y visualizar como procesa el sistema los datos tipo fecha y detectar aquellas dimensiones de y2k (almacenamiento, presentación y aritmética) involucradas.

El *Análisis General* se debe realizar prestando particular atención a la existencia de documentación de diseño de los sistemas (Diagramas de Flujo y DFD's) y, especialmente, de planes, procedimientos y casos de test previamente desarrollados.

Como resultado del *Análisis General* se obtendrá una cuantificación estimada del esfuerzo requerido, esencialmente, en términos de tiempo y recursos (Personal, HW, SW, etc.) y un catastro de requerimientos globales que podrían permitir definir prioridades para la planificación, desarrollo y ejecución del testing. Adicionalmente, si existiera, se adjuntará la documentación de planes de test previamente desarrollados.

Análisis detallado.

Una correcta planificación del proceso de Verificación y Validación para el año 2000 debe contemplar la especificación y/o definición de parámetros como: edad del código, tamaño del sistema y de los programas, complejidad de éstos, número y tipo de interfaces, número y tipos de almacenes de datos, etc.

El resultado del *Análisis Detallado* será un *inventario detallado* que incluirá un catastro de todos los programas componentes del sistema indicando su complejidad respectiva medida en términos de la cantidad o porcentaje de líneas impactadas (idealmente ambos valores), la naturaleza de tal impacto y los demás factores pertinentes a cada programa mencionados anteriormente descritos cualitativa o cuantitativamente.

En el caso de la complejidad, se sugiere una descripción cuantitativa que incorpore tanto la cantidad de líneas impactadas como el total de líneas del programa (tamaño del programa) y el porcentaje respectivo

Usar un porcentaje de impacto para describir la complejidad por sí solo es ambigüo puesto que para dos programas de diferente extensión con igual porcentaje se podría concluir que requieren el mismo esfuerzo de testing, lo cual es absolutamente erróneo. Ahora bien al registrar también el tamaño del programa (o total de líneas del programa), se alcanza una mejor comprensión de la complejidad superando el inconveniente anterior, no obstante, se adolece de certeza, lo cual se puede mejorar agregando una descripción del tipo de impacto presente. Así por ejemplo, para una conversión utilizando la técnica del año límite, en que los almacenes de datos no sufrirán cambios, aquellos segmentos impactados que involucran la dimensión de almacenamiento tendrán una menor complejidad que aquellos que involucran aritmética.

El concepto de complejidad acuñado se adecua mucho mejor al problema del año 2000 que las medidas tradicionales de complejidad, más adecuadas al desarrollo de software y muy poco relacionadas con la mantención de éste. La métrica ciclomática de McCabe, por ejemplo, útil para el desarrollo de sistemas altamente estructurados, no da una medida significativamente representativa de la complejidad de mantención y testing para y2k.

El Análisis Detallado se puede apoyar en la documentación de diseño de los sistemas (si existiera) y especialmente en los diagramas de flujo de los programas.

Identificación de restricciones

Esta fase tiene por objetivo identificar aspectos que pueden constituir limitaciones para la formulación del plan de test que afecten al proceso de testing, a modo de ejemplo, se debe anticipar recortes presupuestarios, prever una disminución de personal y/o evaluar si el ambiente de testing tendrá la capacidad computacional suficiente para implementar el plan de test sin dificultades.

Entre las restricciones más descollantes se pueden mencionar: requerimientos y disponibilidad de recursos humanos, disponibilidad del sistema, requerimientos y disponibilidad de equipos y otras restricciones que surjan del análisis global o del análisis detallado.

Generación del plan global de test

Para terminar la etapa de planificación se debe desarrollar un plan de test global que incluya la *estrategia de testing* a utilizar, la definición del *ambiente de testing, recursos humanos, costos* y una *programación* de las actividades del testing.

Estrategia de Testing Botton-up.

La estrategia de testing Botton-up se inicia a nivel de componentes o unidades elementales que se integran recursivamente construyendo jerarquías o *niveles de integración* hasta llegar al testeo del sistema en su globalidad.

El testing de aplicaciones convertidas para el año 2000 debe utilizar esta estrategia. De este modo, módulos individuales de software serán testeados independientemente antes de iniciar el testing de integración y de sistema, definiendo agrupaciones lógicas de programas y sistemas que interactúan entre sí, las cuales serán testeadas como una entidad única en un escenario determinado.

Subyace a la recomendación anterior la idea de *testear junto al convertir* cada unidad de programa (asegurando primero, el correcto funcionamiento de éstos) y sólo entonces, comenzar la integración hasta alcanzar el testing de sistema, lo cual es más eficiente, puesto que se puede prescindir del uso de drivers, y aún cuando fuera necesario incorporarlos al testing, éstos ya existen, sólo que no han sido convertidos para cumplir con el año 2000 y bastará con tener presente esta restricción en el proceso de testing.

Otra razón por la cual es aconsejable esta estrategia es que se enfoca todo el esfuerzo en convertir y mejorar código que efectivamente será utilizado a posteriori por el sistema en explotación, evitándose la tarea de construir módulos fantasmas (con igual interfaz a la del sistema) que serán desechados al terminar el proceso de conversión.

En estricto rigor la estrategia de testing Botton_up considera los siguientes tipo de test: test de unidad, test de módulo, test de subsistema, test de sistema, test de integración, test de aceptación y test de regresión. No obstante, cada proyecto año 2000 requiere una minuciosa evaluación de los tipos de test a ejecutar.

Ambiente de testing

Se deben identificar, y si fuese necesario adquirir, aquellos componentes ambientales requeridos para el correcto desarrollo y ejecución del testing tales como requerimientos de hardware, software, sistemas operativos, herramientas de test, etc. los cuales deben incluirse en la documentación del plan de test, que constituye el resultado de esta etapa.

Programación del testing

En el plan global se deben identificar todas las actividades globales (o al menos las más relevantes) a desarrollar en el proceso de testing asociadas al tiempo que ellas requieren y reflejarlas en un cronograma o carta Gantt de tal forma de permitir una mejor administración del testing. Esta programación no requiere la inclusión en detalle de todas las etapas de desarrollo del test, pues aún no se ha generado un plan detallado que incluya el desarrollo de procedimientos y casos de test. Por esta misma razón la programación podría someterse a ajustes después de la etapa de desarrollo del testing.

Recursos humanos y costos

Se debe incluir en el plan global aspectos presupuestarios derivados de la estructura de costos para el testing de y2k y de los montos asignados para abordar el proyecto de conversión. Paralelamente, para definir el equipo de testing, se debe exhibir una descripción de los participantes involucrados, el perfil de ellos y la respectiva responsabilidad que asumirán durante el proceso de testing (por ejemplo, usuarios finales para el test de aceptación, asesores externos para la planificación y capacitación, etc.).

Desarrollo del testing

El propósito de esta fase es desarrollar un plan de test detallado precisando tipos, casos y procedimientos de test junto a la definición de niveles de confianza que sirvan para establecer criterios de término del testing.

Las tareas comprendidas en la etapa de desarrollo del testing son:

- Evaluación de los tipos de test a ejecutar.
- Desarrollo de casos y procedimientos de test.
- Definición de niveles de confianza.

Evaluación de los tipos de test a ejecutar

La primera tarea a realizar por el equipo de testing en la etapa de desarrollo es la evaluación de los tipos de test pertinentes a la estrategia de testing botton-up que se aplicarán para buscar errores relativos al problema del año 2000, los cuales se describen a continuación.

Test de Unidad.

El test de unidad no se aplica para ejercitar la interacción entre aplicaciones. Muy por el contrario, los tests de unidad son ejecutados autónomamente sobre componentes de bajo nivel (de acuerdo a la estrategia de testing botton-up mostrada en la figura 2.1) en búsqueda de errores y/o para confirmar que los esfuerzos de conversión han producido los resultados esperados para éstos.

El componente de más bajo nivel a testear, el programa, debe ser sometido tanto a tests estructurales (de caja blanca) como funcionales (de caja negra).

Testing Estructural de Unidad

Para el caso del *testing estructural de unidad* lo más apropiado es realizar *un test de flujos de datos*, seleccionando caminos de prueba a través del programa de acuerdo con la ubicación de las definiciones y manipulación de variables de tipo fecha. El proceso de generación de casos de test se debe apoyar en la documentación del inventario detallado del programa generado en la etapa de planificación del testing.

El testing estructural de los programas debe orientarse a la búsqueda de las siguientes condiciones de error:

- Año en dos dígitos sin uso de rutinas de fecha.
- Sort interno o merge interno indexado por fecha con año en dos dígitos.
- Sort externo.
- Uso de flags con fecha o uso de fecha en código duro.
- Uso de Accept Date.
- Comparación de fecha con dos dígitos sin uso de rutina estándar.

- Inexistencia del 29 de febrero en el calendario del año 2000.

El *año en dos dígitos* sin uso de rutinas de fecha implica que no se ha realizado adecuadamente la conversión, es decir, se trata de casos de no adhesión a los procedimientos de conversión que, simplemente, fueron olvidados al momento de las modificaciones. En el siguiente segmento de programa puede verse que éste es el caso de la variable fech-cb.

```
.

.

007300 WORKING-STORAGE SECTION.

.

.

012300 01 TITULO2-CB.

012400 05 FILLER PIC X(32) VALUE SPACES.

012500 05 FILLER PIC X(40) VALUE

012501 "INFORME ANUAL INVENTARIO ACTUALIZADO AL ".

012700 05 AA-CB PIC 99/99/9999.

012800 05 FILLER PIC X(18) VALUE ", POR CODIGO BIEN.".

012900 05 FILLER PIC X(22) VALUE SPACES.

013000 05 FECH-CB PIC 99/99/99.

013100*

.

.
```

(Fuente: Programa BINPR59 de un sistema de administración de inventarios de la DPI de la Universidad de Concepción convertido para alcanzar su compatibilidad con el año 2000.)

Un *sort interno o merge interno* indexado por fecha con año en dos dígitos corresponde a la utilización de mecanismos de ordenamiento o combinación implementados en el código fuente con el uso de primitivas del lenguaje (por ejemplo, la sentencia SORT de Cobol) formando, entonces, parte del código de la aplicación.

Un *sort externo* es un sort que se utiliza fuera de un programa o código fuente (habitualmente se vende como un producto de software), es decir, al momento de necesitar ordenar una secuencia, el programa

se detiene llama a la pieza de software que ordena y luego vuelve, este caso constituye una condición de error si la rutina externa no es compatible con el año 2000.

El *uso de flags con fechas* se refiere a sistemas que dan una interpretación especial a instancias o valores específicos de los datos tipo fecha. Los más usados son típicamente 00 ó 99 para datos nulos, último registro, fecha infinita, etc.

Código en duro en general significa cualquier codificación a través de una constante no paramétrica. En el mismo sentido, el *uso de fecha en código duro* se refiere al uso del siglo en duro y significa que una aplicación maneja la fecha con cuatro dígitos para el año, pero los primero dos dígitos ("19") son utilizados como una constante.

Para algunas versiones de Cobol (Vax o IBM) el verbo *Accept Date* retorna la fecha de sistema operativo con dos dígitos para el año, es por esto que el uso de Accept Date debe ser considerado una condición de error. Esta situación puede verse en el siguiente segmento de código.

```
.

.

007300 WORKING-STORAGE SECTION.

.

.

008400 01 FECHA1.

008500 05 MM PIC 99.

008600 05 DD PIC 99.

008700 05 AA PIC 99.

.

.

033100 PROCEDURE DIVISION.

.

.

033300 ACCEPT FECHA1 FROM TODAYS-DATE.
.

.
```

(Fuente: Programa MINICOMP de un sistema de administración de inventarios de la DPI de la Universidad de Concepción convertido para alcanzar su compatibilidad con el año 2000.)

La *comparación de fechas con año en dos dígitos* sin uso de una rutina estándar para éstas (que sea compatible año 2000) es una condición de error que, al igual que el sort y el merge indexado por fechas, se genera puesto que lógica o matemáticamente, 00 es menor que 99, lo cual no es semánticamente correcto en el dominio de datos tipo fecha.

Las *comparaciones de fechas* son muy comunes en sistemas financieros y contables. Para este fin, típicamente, se hace uso de un formato del tipo AAMMDD , que es fácilmente ordenable y comparable sobre la base de comparación de cadenas de caracteres.

La *inexistencia del 29 de febrero en el calendario del año 2000* es una condición de error típicamente generada por una rutina incorrecta para la determinación de un año bisiesto.

Testing Funcional de Unidad

Por su parte, el *testing funcional de unidad* se orientará a la verificación de la correcta operación del programa ejercitando además la interacción con los distintos almacenes de datos y en general sus interfaces de entrada/salida.

El testing funcional se orientará a la ejercitación de las fechas críticas mostradas en la tabla 1.

Fecha	Motivo
9 de septiembre de 1999	9999 en el calendario Gregoriano. 9999 denota el fin de entrada en muchos programas.
31 de Diciembre de 1999	Último día del año 1999
1 de enero de 2000	Primer día del año 2000
3 de enero de 2000	Primer día hábil del 2000
10 de enero de 2000	Primera fecha que requiere campos de siete dígitos para su almacenamiento
31 de enero de 2000	Primer fin de mes del año 2000
29 de febrero de 2000	Prueba de año bisiesto
01 de marzo de 2000	Prueba de año bisiesto
31 de marzo de 2000	Primer fin de trimestre del año 2000
30 de junio de 2000	Primer fin de semestre del año 2000
10 de octubre de 2000	Primera fecha que requiere campos de 8 dígitos para su almacenamiento.
31 de diciembre de 2000	Fin del año 2000
1 de enero de 2001	Inicio del año 2001
29 de febrero de 2001	Fecha invalida puesto que 2001 no es un año bisiesto
31 de diciembre 2001	Chequear que el año tiene 365 días

Tabla 1. Fechas Críticas de interés para el testing funcional de unidad.

(Fuente: Mardon Century Experts, Inc. "Year 2000 test Dates Selection Decision Proces Criteria and Process")

Test de integración

Los tests de integración son ejecutados sobre múltiples componentes y aplicaciones para confirmar si el funcionamiento e interacción de éstos con otros programas es apropiado. Luego, para el caso de y2k se aplicarán test que ejerciten la interfaz entre aplicaciones, subsistemas y sistemas que se traspasen fechas, ya sea directamente o a través de un almacén de datos.

Test de regresión

El test de regresión se ejecuta para verificar y validar que el proceso de mantención de una pieza de software no genera nuevos errores en donde no los había. Para y2k, el testing de regresión debe ejecutarse tanto para verificar que el proceso de conversión de los sistemas no genera esta clase de errores como para verificar los sistemas en una fase posterior a la depuración.

Tanto el test de integración como el test de regresión se deben llevar a cabo con el uso de técnicas de caja negra.

Test de sistema

Uno de los últimos pasos en el proceso de testing es validar que el procesamiento u operación del sistema produce los resultados adecuados cuando todas las aplicaciones operan en conjunto.

Los aspectos más importantes que inevitablemente se deben testear son: que el sistema acepte y procese fechas anteriores al siglo 20, que acepte y procese fechas posteriores al siglo 21 (ambos siglos inclusive) y que se reconozca al año 2000 como año bisiesto (ver en la sección 1.6 la definición de compatibilidad con el año 2000).

En este escenario, resulta natural llevar a cabo el test de sistema utilizando la técnica de caja negra (test funcionales) de clases de equivalencia. Son dos las clases de equivalencia relevantes: una que incluya el rango de fechas anteriores al año 2000 y otra para el rango de fechas posteriores a éste.

Esta técnica debe ser reforzada utilizando tests de valores límites, en cuyo caso, las fechas que obligatoriamente deben testearse son el 01 de enero y el 29 de febrero del año 2000 y sus vencindades.

El test del sistema debe contemplar una adecuada planeación del envejecimiento de datos, esto es, simular conjuntos de fechas antes, en y después del año 2000 en almacenes de datos e interfaces entre programas (tarea en sumo dificil).

Test de aceptación.

Finalmente, al llegar al testing del sistema, se hace imprescindible la incorporación del usuario final, pues es la única forma de validar el funcionamiento global del sistema y de añadir cobertura al testing (ver definición de niveles de confiabilidad).

Desarrollo de casos y procedimientos de test

Una vez completada la evaluación de los tipos de test se procederá al diseño de casos de test y al desarrollo de procedimientos que especificarán los casos de test individuales y los pasos que se deben seguir al ejecutar cada caso y tipo de test.

Los casos de test deben diseñarse acorde a cada tipo de test a desarrollar y de modo tal que se

ejecuten todas las funcionalidades del sistema relacionadas con la fecha ejercitando interfaces, almacenes de datos y lógica interna de las aplicaciones (las tres dimensiones de y2k).

Recuérdese que un caso de test exitoso es aquel que encuentra errores, por tanto, la formulación de casos de test debe desarrollarse buscando encontrar la mayor cantidad de errores por test ejecutado con una filosofía desctructiva más que con el afán de comprobar que el sistema procesa las fechas correctamente.

Definición de niveles de confianza

Como parte del plan detallado debe incluirse una definición de los niveles de confianza que se pretende alcanzar para el testing de cada aplicación.

Un bajo nivel de confianza se puede alcanzar minimizando los casos de test funcionales exhibidos en la tabla 1, prescindiendo de los test de regresión e integración y omitiendo la participación del usuario en el test del sistema.

Un nivel medio de confianza estaría definido por la incorporación de tests de integración y aceptación al caso anterior.

Finalmente, un alto nivel de confianza se logrará diseñando casos de test funcionales que ejerciten todo el catastro de fechas de la tabla 1, aplicando todos los tipos de test mencionados en la etapa de evaluación de los tipos de test y permitiendo al usuario incluir valores excepcionales, especificados aleatoriamente, para verificar datos de fecha, detectar cualquier fecha "errónea" y chequear qué valores aparecen en cada campo de fecha agregando, de este modo, una mayor cobertura al testing.

Cada caso de test ejecutado para buscar errores relativos a y2k involucra costos, recursos humanos y tiempo. Si bien, en este sentido, un nivel de confianza alto requerirá de un mayor esfuerzo, es inevitable perseguir el objetivo de mayor confianza con todas aquellas aplicaciones de misión crítica, lo cual se puede relajar tratándose de sistemas no críticos.

Ejecución del testing

La etapa de ejecución del testing tiene por propósito la realización del plan global y detallado, es decir, en esta etapa se aplicarán los casos de test diseñados para cada tipo de test, para lo cual los ejecutores del testing se ceñirán a los procedimientos establecidos en la fase de desarrollo del testing.

Las tareas de la etapa de ejecución del testing son:

- Establecimiento del ambiente de testing
- Capacitación
- Aplicación del Testing

Establecimiento del ambiente de testing.

Se debe implementar el ambiente de testing definido en la etapa de planificación. Por ejemplo, se instalarán aquellos componentes hardware y software identificados en la definición del ambiente de testing del plan global situándolos en condiciones de soportar el testing de las aplicaciones.

Capacitación.

Es esencial desarrollar actividades de capacitación para el personal involucrado en el proceso de testing con el propósito de contrarrestar problemas por falta de entrenamiento o por desorientación de los participantes y, a la vez, maximizar el rendimiento del esfuerzo de testing.

Aplicación del Testing

Esta fase corresponde a la ejecución del testing propiamente tal, esta tarea se debe realizar según lo estipulado en los planes globales y detallados, prestando particular atención al envejecimiento de datos y al comportamiento de los sistemas en el ambiente que se ha implementado bajo la simulación de fechas de los siglos 20 y 21.

Reportes del Testing

El propósito de esta etapa es generar o desarrollar la documentación del proceso y de los resultados del testing y llevar a cabo el posterior análisis de éstos mismos identificando errores y evaluando su impacto. Nótese, entonces, que los reportes del testing han de ser de dos tipos: reportes de los resultados del testing y reportes del proceso de testing.

Análisis y reportes de los resultados del test

Al concluir cada caso de test se debe generar el reporte pertinente que incluya los resultados del test, tales como tipo y número de errores encontrados, número de ejecuciones, tiempo y personal requerido, etc.

Subsecuentemente se debe desarrollar un análisis de los resultados arrojados por el test que pueden incluir factores estadísticos acerca de la aplicación del test, cuyo estudio apoyará la toma de decisiones del equipo o del jefe del equipo de test.

Esencialmente se tendrán warnings, errores de no adhesión a los procedimientos de conversión y errores propiamente tales.

Reportes del Proceso de Testing

La *documentación del proceso de testing* debe incluir el plan global y el plan detallado, precisando los diferentes eventos del test, los puntos de control realizados, el entorno HW y SW utilizado, una descripción de las herramientas de las que se haya dispuesto, casos y procedimientos de test aplicados, conjuntos de fechas testeadas, nivel de confianza alcanzado, etc.

Como planteara Dijkstra en los años 60, el testing puede ser usado para mostrar la presencia de errores, pero nunca su ausencia. Consecuentemente a este principio, el testing para y2k no garantiza por sí solo compatibilidad o cumplimiento con el año 2000, sólo demuestra que ya no se encuentran más errores.

Debido a esto, los reportes del proceso de testing son de vital importancia para la certificación de compatibilidad con el año 2000.

Más aún, como la especificación de criterios de rendimiento y éxito del testing para y2k resulta muy laboriosa, puesto que se adolece de experiencia previa, la única forma de proporcionar certificación es mediante la auditoría del proceso de testing y de los resultados del mismo. Es por esto que los reportes del proceso de testing para y2k deben constituirse en una documentación auditable preparada particularmente para este fin.

Conclusión

Jamás en la historia de la industria informática se había presentado un momento en que una inmensa cantidad de software estuviera siendo examinado, cambiado y testeado a la vez. Es por esto, que el testing será un factor crítico para asegurar que el trabajo y el esfuerzo de conversión han sido efectivos, es decir, " el éxito de un proyecto año 2000 *será puesto a prueba* en la etapa de testing".